


Implementing MCAPI in your Embedded Multicore Application

December 17, 2009
Presented by: **Sven Brehmer**, Chairman of the MCAPI Working Group
sven.brehmer@polycorsoftware.com

Don't let the multicore revolution leave you behind! www.multicore-association.org



Welcome to “Implementing MCAPI in your Embedded Multicore Application”

Presenter Background

► Sven Brehmer

- CEO and founder PolyCore Software, Inc.
- MCA Board member
- Chairman of MCAPI working group
- Significant contributor on MRAPI specification



PolyCore Software® and the PolyCore Software logo are trademarks of PolyCore Software, Inc. All other product or service names are the property of their respective owners. © PolyCore Software, Inc. 2008.

Agenda

- ▶ The Multicore Association
- ▶ MCAPI – Multicore Communications API
 - Overview
 - MCAPI 1.1 sneak preview
- ▶ MCAPI strategies
- ▶ MCAPI case study
- ▶ Conclusions

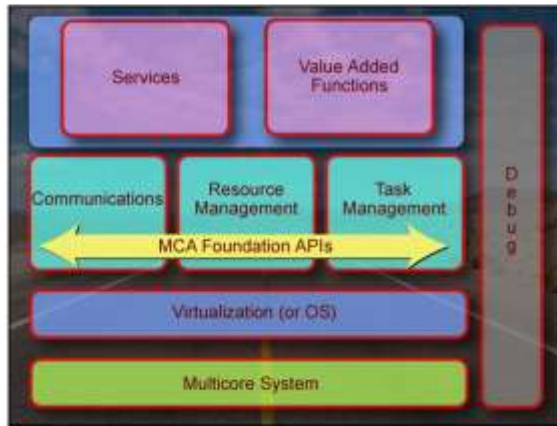
In this webinar, we will cover the following topics:

We will begin with a brief overview of the MCA and its main goals. Then we will go into some detail on MCAPI, including features and benefits.

We will go over some of the new features that will be part of MCAPI 1.1, basically things to increase efficiency and performance. This will be followed by discussing some implementation strategies and tying this to some case studies showing how to optimize the multicore topology for the application.

The Multicore Association

- ▶ Purpose: Forum to address multicore standardization issues.
- ▶ Goal: Improve time to market for applications through the use of standards.



MCA Foundation API's

Communications (MCAP)

- Lightweight messaging

Resource Management (MRAP)

- Basic synchronization
- Shared Distributed Memory
- System Metadata

Task Management (MTAP)

- Task lifecycle
- Task placement
- Task priority

Don't let the multicore revolution leave you behind!

www.multicore-association.org

THE
Multicore
ASSOCIATION

The Multicore Association was founded in May 2005, back in the pioneering days of multicore technology. The primary goal of the MCA is to develop an extensive set of application programming interfaces (APIs) and the establishment of an industry-supported set of multicore programming practices and services. The current MCA API effort supports multicore communications, resource management, task management, and virtualization technology. These APIs are designed to work together or to be used separately.

These and future industry-standard APIs will provide a foundation for a multitude of services and functions including load balancing, power management, reliability, and quality of service, elements, that are also on the consortium's roadmap. Software development tools will be able to take advantage of the unified APIs and standards, independent of specific multicore systems, and will provide support for other MCA roadmap elements such as programming languages and models, hypervisors, design environments, and application generators. Ultimately, the MCA standards will benefit the system developers, whose applications can take advantage of these universally-shared APIs.

Multicore Association Membership

► Executive Board Membership

- Freescale Semiconductor, Huawei Technologies, IBM, Intel, Mentor Graphics, Nokia Siemens Networks, Plurality, PolyCore Software, Samsung Electronics, Texas Instruments, Tiler, Wind River

► Working Group Membership

- CAPS entreprise, Codeplay, CriticalBlue, Enea, eSOL, IMEC, LG Electronics, LSI, MIPS Technologies, National Instruments, nCore Design, Open Kernel Labs, Siemens, Virtutech

► University Membership

- Carnegie Mellon University, Delft University of Technology, University of Houston, University of Utah

Be Part of the Multicore Revolution

www.multicore-association.org

Don't let the multicore revolution leave you behind

www.multicore-association.org

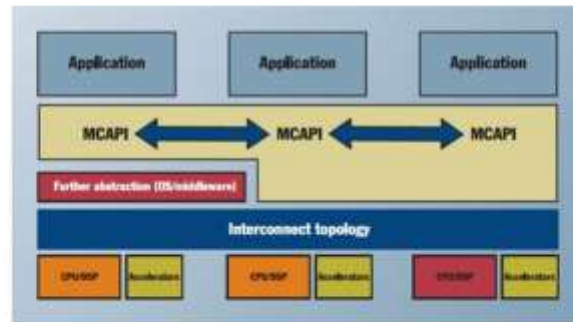
THE
Multicore
ASSOCIATION

The membership is comprised of Executive Board, Working Group, and University Membership. The current list of members spans a wide area of disciplines ranging from software vendors to processor vendors to system developers, and universities working on a range of multicore-related projects. (NOTE: Currently, all Executive Board member positions are filled)

MCAPI – Addressing the Need

Purpose: Define a standard communications API for closely distributed systems

- ▶ Previous multiprocessing standards address communications for widely distributed applications

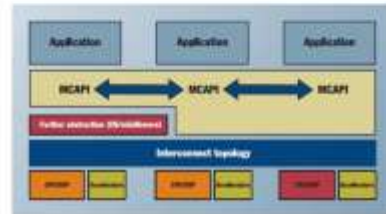


Prior to MCAPI, there was no multicore API that could efficiently address the need for a closely distributed system. Previous multiprocessing standards, such as OpenMP and MPI, are targeted at widely distributed systems.

MCAP1 – Addressing the Need

Objectives

- ▶ Source level portability
- ▶ Lightweight, performance capable
- ▶ Scalable to thousands of cores
- ▶ Simple, extendable



MCAP1 Domain:

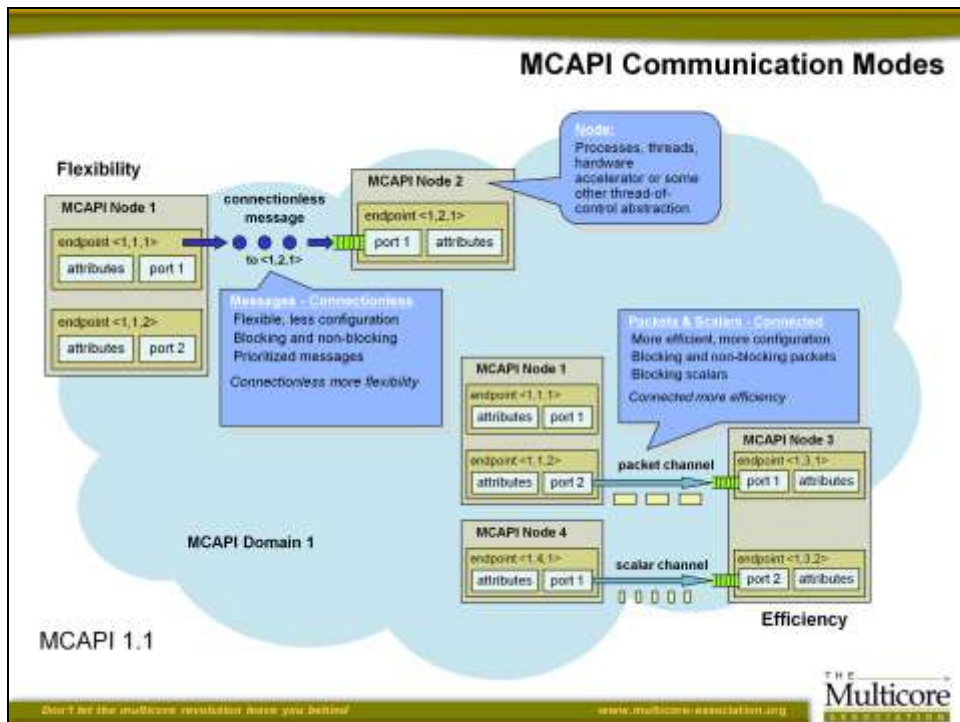
- ▶ Closely distributed applications (board level – multicore, multi-chip)
- ▶ Resource constrained systems with real-time requirements
- ▶ Static topology, reliable transports
- ▶ Multitude of cores, OS'es & interconnects

It's hard to keep it simple and not throw in the kitchen sink. It also had to be extensible to allow more functionality to be layered on top.

The MCAP1 domain is closely distributed systems and that means board level, where you have multiple cores on a chip or multiple chips on a board, or any combination thereof. And typically resource constrained systems with real-time requirements, which is common in embedded systems.

This is a static topology, meaning that the same number of cores are there every time you turn on the power switch. Transports are reliable because they should be there every time.

This could include multiple types of cores, such as CPUs and DSPs, and a different number of cores. It could also include multiple instantiations of the same operating system or different operating systems. It also could include a variety of interconnects, such as shared memory and other things.



Talking about some of the fundamentals of MCAPI.

Nodes are very important in MCAPI. They were defined as a thread of control which can be a process, a thread, a hardware accelerator, or something else. We did this because we wanted MCAPI to be implementable in a wide variety of multicore platforms.

MCAPI domain, is something that will be introduced with MCAPI 1.1, is used for routing purposes.

A port is the finest grain identifier of an endpoint.

There are 2 modes of communication: 1) connectionless messages, these are the most flexible and slightly less performant; 2) connected channels, included both packets and scalars, and these are meant for more performant communications.

MCAPI - Concepts

- ▶ **Topology Management**
 - Endpoint = <domain, node, port>
 - Initialization
 - Create/get endpoints
 - Attribute get/set
- ▶ **Messages - Connectionless**
 - Per message priority
 - Blocking & non-blocking
 - User provided buffers send & recv
 - Availability
- ▶ **Packet Channels – Connected**
 - FIFO
 - Blocking & non-blocking
 - Send/user & recv/"system" provided buffers, buffer release
 - Connect, open & close
 - Availability
- ▶ **Scalar Channels – Connected**
 - FIFO
 - 64, 32, 16 and 8 bit scalars
 - Connect, open & close
 - Availability
- ▶ **Non-blocking management**
 - Test, wait and cancel

Don't let the multicore revolution leave you behind! www.multicore-association.org

THE Multicore ASSOCIATION

There are 5 different functional areas of MCAPI.

1) Topology management: involves initializing an implementation, creating and deleting endpoints, getting endpoints (explained later), and allows an application to get and set endpoint attributes.

2) Connectionless messages: messages are buffered communication and they allow per message priority. They can be issued in blocking and non-blocking fashion. We will discuss the significance of this in a few slides. The buffers used in a communication are user provided on both the send and receive sides. It's also possible to check to see how many messages are waiting to be read on an endpoint.

3) Packet channels, as a connected communication, are unidirectional channels. They have FIFO order. This is also buffered communication and can be in blocking and non-blocking fashion. Different from the messages is that for packet channels, the buffers are user provided on the send side and implementation provided on the receive side; this is to facilitate zero-copy communication. Packet channels also include some functionality to manage the buffers. Channels also require opening and closing.

4) Scalar channels are aimed at systems that have hardware support for sending small amounts of data (for example, a hardware FIFO). There are four flavors including 64, 32, 16, and 8 bits. These only have blocking functions because they are meant to be only performant.

5) For non-blocking calls, the application receives a token for each request and can then use the non-blocking management function to test if the request has completed, wait for it (either singularly or wait for any one of requests in an array of requests). They can also be cancelled.

Getting started with MCAPI

Can be as simple as:

Manage

```
mcapi_initialize(params) ;  
mcapi_endpoint_create(params) ; /* local */  
mcapi_endpoint_get(params) ; /* remote*/  
mcapi_finalize(params) ;
```

Communicate

```
mcapi_send(params) ;  
mcapi_recv(params) ;
```

We strived to keep MCAPI simple. While most MCAPI users will use more than these 6 functions, you can get started with MCAPI using only these.

- ▶ Initialization, setup and discovery
- ▶ Load balancing
- ▶ Performance

Let's discuss some strategies that you can use with MCAPI in your application.
We will discuss initialization, setup, and discovery.
Load balancing to get the most out of your platform.
Performance.

MCAPI – Node initialization, setup and discovery

Objectives:

- Balanced performance and resources
- Ability to adapt application to different # of cores

► Node initialization (each nodes is initialized)

- Parameterize for balanced performance and resources

- `mcapi_initialize(domain_id, node_id, (void *)&mcapi_parameters, &mcapi_info, &mcapi_status);`
 - Implementations specific parameters, which may include pre-allocated resources

- Utilizes initialization information for initial setup and discovery

- `mcapi_initialize(domain_id, node_id, (void *)&mcapi_parameters, &mcapi_info, &mcapi_status);`
 - Including number of topology nodes (static topology) and available ports

The objective is to balance performance with a reasonable amount of resources.

It's also preferable if the application can adapt to more than one topology. This will be discussed further.

The first thing that you'll do is initialize each node. The concept of MCAPI parameters is new in MCAPI 1.1. It allows an application to parameterize the implementation and gives you the ability to manage, for example, pre-allocated resources to use only what you need.

In the other direction, the initialization can provide information about the topology (such as number of available nodes). It can also tell you the number of available ports on your endpoints.

MCAPI – endpoint setup and discovery continued

► Endpoint setup

- Create endpoints on local node (# of ports)

```
for(i = 0; i < mcapi_info.number_of_ports; i++){
    endpoint[NODE_1][i] = mcapi_endpoint_create(port_id[i], &mcapi_status);
    if(mcapi_status != MCAPI_SUCCESS) break;
}
```

► Topology discovery & setup: Get endpoints from other nodes (# of nodes)

```
for(i = 0; i < mcapi_info.number_of_nodes; i++){
    if(i != my_node_id){
        endpoint[i][0] = mcapi_endpoint_get(domain_id, node_id[i], port_id[0], &mcapi_status);
        if(mcapi_status != MCAPI_SUCCESS) break;
    }
}
```

- Channel connections can be setup collectively (known topology) or individually by the nodes after discovery

► Capability discovery

- Identify specific node capability
 - Functionality: FFT, packet inspection, codec, etc
 - Characteristics: resources, performance and more
 - mcapi_msg_send() & mcapi_msg_rcv()

Don't let the multicore revolution leave you behind!

www.multicore-association.org

THE
Multicore
ASSOCIATION

After initialization, you need to set up the endpoints (endpoints are the terminating points for communication) on each side of the communication.

Start by creating endpoints on your own node. For example, you could use one for each of the available ports.

You also need to get endpoints from other nodes so you can communicate with them. By using the information about the number of nodes, you could get an endpoint from each other node and use that in your communication.

Channels can be set up collectively if the topology is known or individually at runtime by each node.

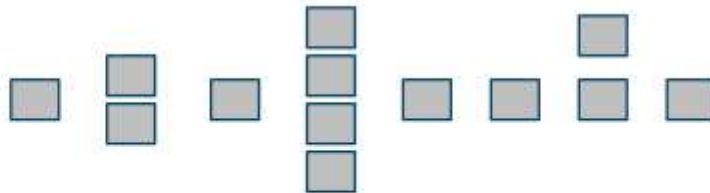
The topology discovery would tell you how many nodes there are in the system. You can then do a second level discovery about the capabilities of your system; this is done at the application level. This could be things such as what functionality is available on the specific nodes and what are the characteristics such as resources and performance. This is very convenient by using the message functionality.

► Topology discovery

- Topology A – 9 nodes



- Topology B - 13 nodes



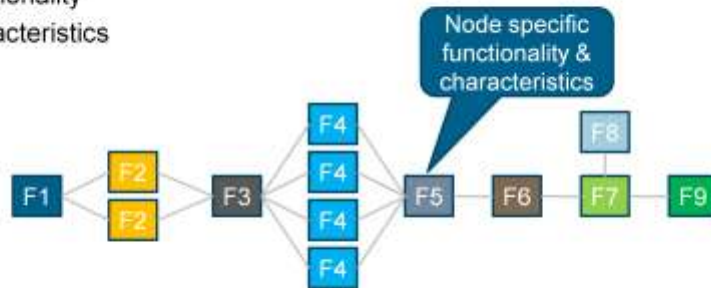
`mcaapi_info.number_of_nodes`

You can do the initial topology discovery by looping through by looping through the number of nodes as you get an endpoint on each node. These examples shows topologies with 9 and 13 nodes. At first, you don't know much about the capabilities of these nodes, unless this was non priori.

MCAPI – Initialization, setup and discovery continued

► Capability discovery

- Functionality
- Characteristics

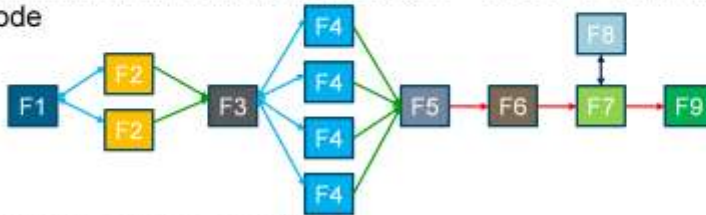


- Application adaptation

In the next step, you can find out specifically what the different nodes can offer (depicted by different colors in the example). Ideally, the application should be able to adapt to the topology, in terms of the number of nodes and their capabilities. If your application is set up correctly, you should be able to change from one topology to another without modifying the source code.

MCAPI – Load balancing

- ▶ Function pipeline with parallel stages – choice of communication mode



- ▶ Multiple senders or receivers

- Messages - Connectionless
 - Unidirectional communication → multiple to one - data
 - Bi-directional communication ↔ one to multiple – data + data requests
 - Bi-directional communication ↔ one to one – data

- ▶ One sender one receiver

- Packet channel
 - Unidirectional communication → one to one - data

Don't let the multicore revolution leave you behind!

www.multicore-association.org

THE
Multicore
ASSOCIATION

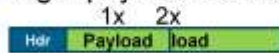
In load balancing, it is important to consider MCAPI's different communication modes. In this example showing a function pipeline with some parallelism (F2 and F4). In cases where there may be multiple senders or receivers it is best to use messages.

You may have uni-directional communication (such as shown between F2 and F3 and between F4 and F5) in which only data flows. You may also have bi-directional communication (such as F1 to F2, F3 to F4) where data flows in one direction and requests for data flowing in the other direction. There may also be bi-directional communication which is one to one (F7 to F8) which is an exception handler.

Packet channels are the best choice if you have uni-directional one-to-one (such as F5 to F6, F6 to F7, and F7 to F9) for higher performance but less flexibility.

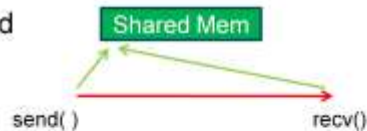
Minimize or “hide” communication overhead

- ▶ Larger payload reduces processing and header overhead



- ▶ Channels reduce processing and potentially transfer overhead

- ▶ Zero copy reduces transfer overhead



- Consider shared memory communication vs. computation

- 2 copies
- 1 copy
- 0 copy



Besides load balancing, there are other things to consider in regards to performance. We want to minimize or hide the communication overhead. It is always the objective in parallel processing to overlap the computing and communication functions.

There is some overhead in the processing of a request and in the header being sent across in the transport. You want to optimize by using the largest payload/header ratio.

As indicated earlier, channels can reduce both the processing and potentially transfer overhead, in that you can set up certain things before you do the connection.

Zero copy reduces transfer overhead and it simply uses shared memory. The sender has a buffer in shared memory and it can pass a pointer to the receiver with no need to physically move the data.

It's important to consider shared memory communication versus the computation.

Demonstrating with three scenarios:

1. Sender that sends from its local memory to shared memory and then to a receiver's local memory. This had a 'cost' of two copies, and while not too efficient the implementation can manage it completely and do all the locking for you.
2. If there's a buffer in shared memory, the sender operates on it and then passes it on to the receiver. This consumes one copy.
3. Zero copy which only passes a pointer.

Minimize or “hide” communication overhead

▶ Non-blocking functions

- Issue communications request and compute until data dependency
 - `mcapi_msg_send_i(); mcapi_wait(mcapi_request);`

▶ Use `mcapi_msg_available` and blocking functions in combination

- If x messages/packets/scalars available, then x read will not block
 - `x = mcapi_msg_available(); for(i = 0; i < x; i++) mcapi_msg_rcv();`

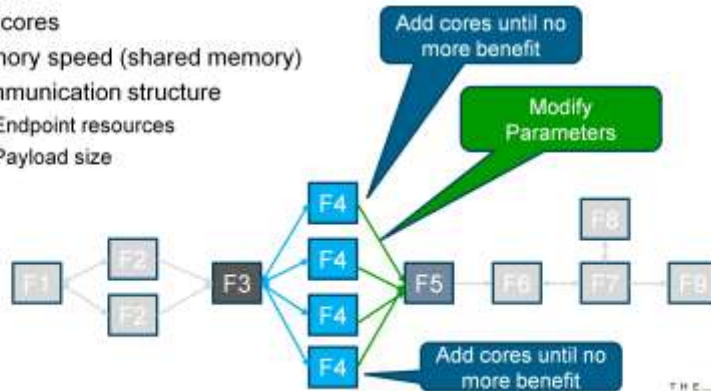
One way to get better performance is to hide or minimize communication overhead.

With non-blocking functions, a request is issued to read some data and then you continue to compute until you have a data dependency (at which point you would do an `mcapi_wait` with your request that was issued during the non-blocking).

You can also use the `mcapi_msg_available` (or packet or scalar) to find out how many are available to be read. So for example, if 3 messages were available **then you can use the blocking read to receive 3 reads without blocking**. This is another way to manage your compute vs communication.

MCAPI case study

- ▶ Packet processing
 - Very compute intensive function **F4**
- ▶ Objective
 - Find out optimal multicore configuration
- ▶ Parameters
 - # of cores
 - Memory speed (shared memory)
 - Communication structure
 - Endpoint resources
 - Payload size



Don't let the multicore revolution leave you behind!

www.multicore-association.org

THE
Multicore
ASSOCIATION

This application was a packet processing application with one very compute intensive function (F4). The objective was to find what would be a good multicore platform.

Parameters at our disposal included:

1. Number of cores
2. Memory speed (specifically for shared memory)
3. With communication, we could change the resources in the topology and the payload size.

We added cores at different memory speeds until there was no more benefit. The objective was to get linear improvement every time we added an F4 core.

Then we started looking at the communication itself, discovering that it might be beneficial to pass multiple packets in each payload.

► Platform

- Simulator (Simics), RTOS (ThreadX), MCAPI communication framework (Poly-Messenger/MCAPI)
- PPC440 – local and shared memory
- Shared memory transport

► Tools

- Topology configuration and generation tools
 - Poly-Mapper & Poly-Generator

► Process

- Add **F4** cores until no more benefit at a certain memory speed
- Change memory speed and try again
- Repeat with multiple packets/payload
- Application unchanged with increasing number of cores
 - Used topology discovery
- Reconfiguration with the tools

We did the experimentation using a simulator (Simics from Virtutech) and an RTOS (ThreadX from Express Logic), and an MCAPI communication framework (Poly-Messenger/MCAPI from PolyCore Software).

The processor was PowerPC PPC440 and each processor had local memory and a shared global memory that was used for the communication.

We used tools for the topology configuration and generation (these were Poly-Mapper & Poly-Generator from PolyCore Software).

There was a one to one mapping between an F4 and a core.

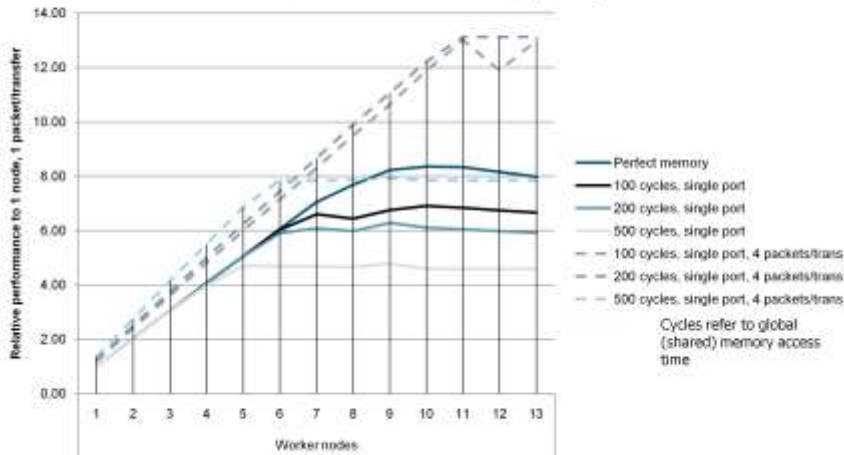
Memory speed was varied and we also used 'perfect' memory to explore the reference.

We repeated with multiple packets per payload.

The application was set up to use the topology discovery to learn how many cores were available to avoid changing the source code between runs. All reconfiguration was done through the tools.

► **Results** – application running on different # of cores and with different payload

Parallel and transfer size speedup



The solid lines pertain to the results of single packets per transfer. Dotted lines are 4 packets per transfer.

With perfect memory, the data shows almost linear performance increase up to 7, and after 9 there was no more benefit.

At 500 cycles, performance topped out at 5 F4 cores.

With larger payload, performance was pushed further all the way up to about 11 F4 cores.

► Case Study Wrap-up

- Difficult to predict the multicore behavior of a certain application
- MCAPI provided a simple and efficient programming model
 - The version 1.1 discovery features proved useful
 - Load balancing was simple with messages
- Rapid prototyping is very useful for identifying a good match between the application and the multicore platform
- The simulator + RTOS + multicore tools combination allowed us to make many iterations in a very short period of time

It's difficult to predict how your application will behave in a multicore system.

MCAPI provided a convenient and simple programming model, the discovery was useful, and load balancing was simple to do with messages. We used the endpoint ID of each F4 core and sent that back on a queue.

Rapid prototyping is useful for identifying how your application will behave and what is the optimal configuration. It is fantastic to be able to do this before committing your application to hardware.

The tools and runtime software allowed us to try many configurations in a very short period of time. This may not be even possible with real hardware.

MCAPI 1.1 sneak preview

- ▶ API consistency, functions and error/status codes
 - API function nomenclature:
 - mcapi_<functional area>_<action>, e.g. mcapi_msg_send()
- ▶ New functions and functionality
 - mcapi domain added
 - mcapi_domain_id_get()
 - mcapi_pktchan_release_test()
 - Initialization parameters and information
- ▶ MCAPI endpoint attribute ranges

Some changes were made to make it more consistent.

The general nomenclature is mcapi prefix followed by a functional area and an action (such as mcapi_msg_send())

We added a domain which is primarily for routing purposes

Added some support for zero copy

Added functionality for parameterization and improved information gathering.

Added capability to have a vendor-specific range of attributes that would have to be requested through the Multicore Association

Take Action

- ▶ MCAPI provides a simple, efficient and consistent multicore programming model across different:
 - Types and number of cores
 - Operating systems
 - Physical transports
- ▶ MCAPI, MRAPI and future MCA standards share concepts and certain data types
 - Easy to use individually and together
- ▶ Relevant standards influence technology adoption
- ▶ Get started, become an influencer
 - Join us at the Multicore Association
 - Contact markus.levy@multicore-association.org
 - For in depth MCAPI information, feel free to contact me

Don't let the multicore revolution leave you behind!

www.multicore-association.org

THE
Multicore
ASSOCIATION

MCAPI provided a simple and efficient programming model to span heterogeneous platforms (in terms of types and number of cores, different operating systems, and physical transports). MCAPI provides transparency.

MRAPI, the resource management API, is coming in spring of 2010. We made some of the aspects of MCAPI and MRAPI the same. For example, they share certain concepts and data types. This will allow MCA standards to be used individually or together.

Relevant standards are essential to make multicore mainstream. The gap between hardware and software is greatest with multicore.

We encourage you to join the Multicore Association.

For more information about MCAPI, or about Poly-Messenger/MCAPI contact Sven Brehmer.

If you have questions

please use the online-question box

Thank you

sven.brehmer@polycoresoftware.com

Questions/Answers

- ▶ How does MCAPI relate to MRAPI, does it depend on it?
 - There are synergies between them, but there are no dependencies. But some of the functionalities of MRAPI, such as shared memory management, will certainly be of help when managing shared memory for MCAPI.

- ▶ Does the implementation expect nodes to act as intermediate hops for sending data. For example, if there are 3 nodes (A, B, C) and A is connected to B and B is connected to C, should A be able to reach C through B?
 - This is more of an implementation question and the answer for Poly-Messenger implementation is yes, you can do multi-hop.

Questions/Answers

- ▶ Can MCAPI conflict with existing communication services in an OS?
 - This should not be the case.

- ▶ Does MCAPI place requirements on the hardware (e.g. hardware semaphore, layout)?
 - Any implementation of MCAPI would probably use the underlying synchronization primitives if they are in the software or hardware.

- ▶ Do you have a roadmap for future revisions of the MCAPI specification (for example, MCAPI header, global database, etc.)?
 - Yes. Originally in the MCAPI 1.0 spec, we identified 3 areas for extension. One was zero copy, which is being done in MCAPI 1.1. Second was multicast, which is for one sender and multiple receivers, and we have not done that but is a likely extension. Third is statistics and debug information, which is also likely to happen.

- ▶ How many vendors are using MCAPI?
 - PolyCore Software has it, and others are working on it but cannot be discussed publicly.

Questions/Answers

- ▶ Can you provide more background on the tools referenced in the presentation's case study?
 - These are tools from PolyCore Software. We used a topology mapping tool; this is a graphical tool that helps you layout and configure a topology. It is model based and generates a topology map that we feed into another tool called Poly-Generator which generates a topology in C.

- ▶ Can we use MCAPI for many-core platforms where number of cores can go up to 64 and 256?
 - Yes, there is no practical limit.

- ▶ Can MCAPI be extended for distributed computing environment, maybe similar to cloud computing standard?
 - This is one area that could be considered for the future.

- ▶ Is there any implementation in sight for MTAPI? So far there are many task-based programming paradigms?

To be clear, MTAPI is not yet available as an API. It follows MRAPI.

Questions/Answers

- ▶ What is the relationship between cores in a multicore processor and MCAPI nodes or endpoints?
 - Node is a thread of control which could one of many things. In essence there can be many nodes on one core or it could be one to one. There is no restriction. You can also have one or more endpoints on a node.

- ▶ Are there any benchmarks for MCAPI?
 - There are in the implementation available online at MCA's website. There are also efforts going on through EEMBC.